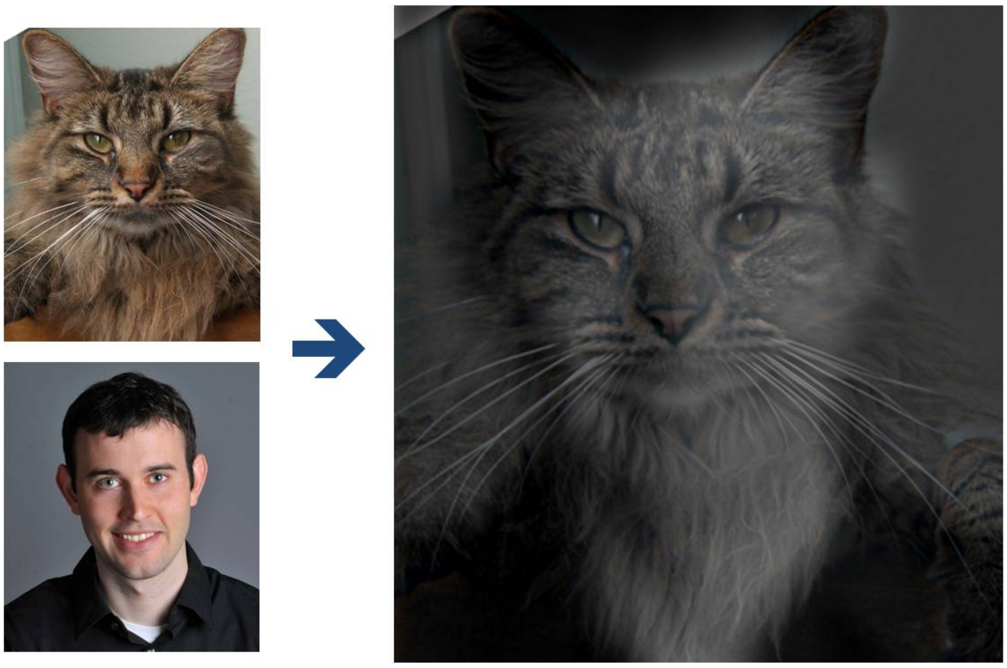
Programming Project #1: Hybrid Images

CS445: Computational Photography



(Look at image on right from very close, then from far away.)

Due Date: 11:59pm on Tues, Sept. 17, 2019

Part I: Hybrid Images

This part of the project is intended to familiarize you with image filtering and frequency representations. The goal is to create hybrid images using the approach described in the SIGGRAPH 2006 paper by Oliva, Torralba, and Schyns. *Hybrid images* are static images that change in interpretation as a function of the viewing distance. The basic idea is that high frequency tends to dominate perception when it is available, but, at a distance, only the low frequency (smooth) part of the signal can be seen. By blending the high frequency portion of one image with the low-frequency portion of another, you get a hybrid image that leads to different interpretations at different distances.

Here, I have included two sample images (of me and my former cat Nutmeg) and some starter code that can be used to load two images and align them. The alignment is important because it affects the perceptual grouping (read the paper for details).

First, you'll need to get a few pairs of images that you want to make into hybrid images. You can use the sample images for debugging, but you should use your own images in your results. Then, you will need to write code to low-pass filter one image, high-pass filter the second image, and add (or average) the two images. For a low-pass filter, Oliva et al. suggest using a standard 2D Gaussian filter. For a high-pass filter, they suggest using the impulse filter minus the Gaussian filter (which can be computed by subtracting the Gaussian-filtered image from the original). The cutoff-frequency of each filter should be chosen with some experimentation (or equivalently choose the sigma/width of the Gaussian). The starter package also includes a `gaussian_kernel` function in `utils.py` to save you some time. Don't just use the `gaussian_filter` function in scipy! Use the result of `gaussian_kernel` as input to a general filter/convolution function.

For your favorite result, you should also illustrate the process through frequency analysis. Show the log magnitude of the Fourier transform of the two input images, the filtered images, and the hybrid image. In Python, you can compute and display the 2D Fourier transform using Matplotlib

and Numpy with:
`plt.imshow(numpy.log(numpy.abs(numpy.fft.fftshift(numpy.fft.fft2(gray_image)))))`

Try creating a variety of types of hybrid images (change of expression, morph between different objects, change over time, etc.). The site has several examples that may inspire.

Part II: Image Enhancement

You may sometimes find that your photographs do not quite have the vivid colors or contrast that you remember seeing. In this part of the project, we'll look at three simple types of enhancement. You can do two out of three of these. The third is worth 10 pts as a bells and whistle.

Contrast Enhancement: The goal is to improve the contrast of the images. The poor constrast could be due to blurring in the capture process or due to the intensities not covering the full range. Choose an image (ideally one of yours, but from web is ok) that has poor contrast and fix the problem. Potential fixes include Laplacian filtering, gamma correction, and histogram equalization. Explain why you chose your solution.

Color Enhancement: Now, how to make the colors brighter? You'll find that if you just add some constant to all of the pixel values or multiply them by some factor, you'll make the images lighter, but the colors won't be more vivid. The trick is to work in the correct color space. Convert the images to HSV color space and divide into hue, saturation, and value channels (`hsv = cv2.cvtColor(im,cv2.COLOR_BGR2HSV)` in OpencCv). Then manipulate the appropriate channel(s) to make the colors (but not the intensity) brighter. Note that you want the values to map between the range defined by the imported library (in OpenCv 0-255), so you shouldn't just add or multiply with some constant. Show this with at least one photograph. Show the original and enhanced images and explain your method.

Color Shift: Take an image of your choice and create two color-modified versions that are (a) more red; (b) less yellow. Show the original and two modified images and explain how you did it and what color space you've used. Note that you should not change the luminance of the photograph (i.e., don't make it more red just by increasing the values of the red channel). In OpenCv use `cv2.cvtColor(image, cv2.COLOR_BGR2Lab)` for converting between RGB and LAB spaces, in case you want to use LAB space.

Bells & Whistles (Extra Points)

- Try using color to enhance the effect of hybrid images. Does it work better to use color for the high-frequency component, the low-frequency component, or both? (5 pts)
- Illustrate the hybrid image process by implementing Gaussian and Laplacian pyramids and displaying them for your favorite result. This should look similar to Figure 7 in the Oliva et al. paper. (15 pts)
- Do all three image enhancement tasks. (10 pts)

Deliverables

To turn in your assignment, place your index.html file and any supporting media in your project directory. On Compass, you will also submit code, a thumbnail, and text outlining how many points you think you should get. See project instructions for details.

Use words and images to show us what you've done (the web page doesn't need to be fancy). Please:

- Show us your favorite hybrid image result. Include: 1) the original and filtered input images; 2) the hybrid image; and 3) the FFT images. Briefly (a few sentences) explain how it works, using the included images as illustrations. Explain any clever ideas that you've incorporated and any parameters. This should be with your own images (not the included samples).
- Next, show us at least two more hybrid image results, including one that doesn't work so well (failure example). Briefly explain how you got the good results (e.g., chosen cut-off frequencies, alignment tricks, other techniques), as well as any difficulties and the possible

reasons for the bad results. If you are so fortunate that everything that you try works well, try to figure out what shouldn't work.
- Do at least two of the image enhancement tasks, showing the original image, the resulting image(s), and explaining how the enhancement was done.
- Describe and bells and whistles under a separate heading.

Scoring

The core assignment is worth **100** points, as follows:

- **45 points** for implementation to create hybrid images from two aligned input images.
- **25 points** for illustration and additional results: 15 points for FFT images; 10 points for including at least two examples beyond the first (including at least one failure).
- **10 points** for quality of results (e.g., 0=poor 5=average 10=great 15=amazing)
- **20 points** for two image enhancement tasks (10 pts each), including explanation and display of results.

You can also earn up to **30 extra points** for the bells & whistles mentioned above (5 for experimenting with color; 15 for Gaussian/Laplacian pyramids; 10 for third task of color enhancement) or suggest your own extensions (check with prof first).

Tips

- You will have learned all the material you need for the hybrid image part by Sept 4, so do that part early. The image enhancement part can be done quickly, but the material is not fully covered until Sept 13.
- Useful Scipy/Numpy/Matplotlib functions: cv2.filter2d, scipy.signal.convolve2d, numpy.ndimage.fft2, matplotlib.pyplot.plt.imshow, matplotlib.colors
- When specifying a Gaussian filter, choose a value for standard deviation and then choose a filter size such that the values near the edges of the filter are near zero
- When displaying FFT images, it may help to specify a scaling range using the 'norm ' argument of `matplotlib plt.imshow(im, norm=LogNorm(vmin, vmax))`
- With high-pass filtered images or laplacian pyramid images, many of the values will be negative. To display them, you need to rescale the images to range from 0 to 1. `numpy.clip` and `image.astype(numpy.uint8)` might be useful.